15-451 — Algorithms — Spring 2006

Sleator, Golovin, Kissner

Homework #2

Due: Monday, February 13, 2006.

Ground rules:

- This is an oral presentation assignment. You should work in groups of three. At some point before **Feb. 12**, your group must sign up for a 1-hour time slot on the signup sheet on the course web page. (The signup sheet will be up on the website by Feb. 7).
- You are not required to hand anything in at your presentation, but you may if you choose. If you do hand something in, it will be taken into consideration (in a non-negative way) in the grading.

Problems:

1. [median of two sorted arrays] Let A and B be two sorted arrays of n elements each. We can easily find the median element in A — it is just the element in the middle — and similarly we can easily find the median element in B. (Let us define the median of $2k$ elements as the element that is greater than $k-1$ elements and less than k elements.) However, suppose we want to find the median element overall — i.e., the nth smallest in the *union* of A and B. How quickly can we do that? You may assume there are no duplicate elements.

   Your job is to give tight upper and lower bounds for this problem.[1] Specifically, for some function $f(n)$,

   (a) Give an algorithm whose running time (measured in terms of number of comparisons) is $O(f(n))$, and

   (b) Give a lower bound showing that any comparison-based algorithm must make $\Omega(f(n))$ comparisons in the worst case.

   In fact, see if you can get rid of the O and Ω to make your bounds *exactly* tight in terms of the number of comparisons needed for this problem.

   Some hints: You may wish to try small cases. For the lower bound, you should think of the output of the algorithm as being the location of the desired element (e.g, "$A[17]$") rather than the element itself. How many different possible outputs are there?

[1] Historical note: this fact — the matching upper and lower bounds — was something that led the [BFPRT] authors to investigate and solve the linear-time median-finding problem discussed in class.

2. [amortized analysis] Suppose we have a binary counter such that the cost to increment or decrement the counter is equal to the number of bits that need to be flipped. We saw in class that if the counter begins at 0, and we perform n increments, the amortized cost per increment is just $O(1)$. Equivalently, the total cost to perform all n increments is $O(n)$.

   Suppose that we want to be able to both increment *and* decrement the counter.

   (a) Show that even without making the counter go negative, it is possible for a sequence of n operations starting from 0, allowing both increments and decrements, to cost as much as $\Omega(\log n)$ amortized per operation (i.e., $\Omega(n \log n)$ total cost).

   (b) To reduce the cost observed in part (a) we'll consider the following *redundant ternary number system*. A number is represented by a sequence of *trits*, each of which is 0, +1, or −1. The value of the number represented by $t_{k-1}, \ldots, t_0$ (where each $t_i, 0 \leq i \leq k-1$ is a trit) is defined to be

   $$\sum_{i=0}^{k-1} t_i 2^i.$$

   For example, | 1 | 0 | -1 | is a representation for $2^2 - 2^0 = 3$.

   The process of incrementing a ternary number is analogous to that operation on binary numbers. You add 1 to the low order trit. If the result is 2, then it is changed to 0, and a carry is propagated to the next trit. This process is repeated until no carry results. Decrementing a number is similar. You subtract 1 from the low order trit. If it becomes -2 then it is replaced by 0, and a borrow is propagated. Note that the same number may have multiple representations (e.g., | 1 | 0 | 1 | = | 1 | 1 | -1 |). That's why this is called a *redundant* ternary number system.

   The cost of an increment or a decrement is the number of trits that change in the process. Starting from 0, a sequence of n increments and decrements is done. Give a clear, coherent proof that with this representation, the amortized cost per operation is $O(1)$ (i.e., the total cost for the n operations is $O(n)$). Hint: think about a "bank account" or "potential function" argument.

3. [A Random Walk] For this problem, you may find Stirling's approximation for $n!$ useful:

   $$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \cdot \left(1 + \frac{1}{12n} + \Theta(1/n^2)\right)$$

   Consider the following random process. A robot is walking along the real number line. Let $p(t)$ be a random variable equal to the robot's position

at time t. It is initially at position $p(0) = n$, and its goal is to get to position one. At each time step t, it moves a distance towards the goal which is uniformly distributed in $\{1, 2, \ldots, p(t)-1\}$. Furthermore, at each time step t, the robot pays a cost of X_t, which is a random variable that is dependent on $p(t)$ and $p(t+1)$ in ways you don't know. You do, however know that $\mathbf{E}[X_t] \leq 2$ for each t.

(a) Give the best upper bound you can on the expected amount the robot pays before reaching it's goal position.

(b) Show how we can use this analysis to bound the expected number of comparisons that are made to the smallest element in an array when we quick-sort the array (using the algorithm in the notes, lecture 3).

(c) Explain why this analysis won't bound the expected number of comparisons that are made to an arbitrary element of the array.

(d) Explain how you could alter this approach to bound the expected number of comparisons that are made to any fixed element in an array when we quick-sort the array. For example, can you give a different random walk on $\mathbb{N}$ that describes the number of comparisons made to a fixed element during the quick-sort execution? How about a random walk on some other set, like $\mathbb{N}^2$ (i.e. $p(t) \in \mathbb{N}^2$ rather than in $\mathbb{N}$)? (We will also accept alternate workable ideas for generalizing this analysis).